Creation of a GUI for Zori, a Quantum Monte Carlo program, using Rappture

Roberto Olivares-Amaya[a], Romelia Salomón-Ferrer[b,c], William A. Lester Jr.[b,c] and Carlos Amador-Bedolla[a]

[a]Departamento de Química Teórica, Facultad de Química,
Universidad Nacional Autónoma de México, México D. F., 04510
[b]Kenneth S. Pitzer Center for Theoretical Chemistry,
Department of Chemistry, University of California
at Berkeley, Berkeley, California 94720-1460
[c]Chemical Sciences Division, Lawrence Berkeley National Laboratory,
Berkeley, California 94720

In their research laboratories, academic institutions produce some of the most advanced software for scientific applications. However, this software is usually developed only for local application in the research laboratory or for method development. In spite of having the latest advances in the particular field of science, such software often lacks adequate documentation and therefore is difficult to use by anyone other than the code developers. As such codes become more complex, so typically do the input files and command statements necessary to operate them. Many programs offer the flexibility of performing calculations based on different methods that have their own set of variables and options to be specified. Moreover, situations can arise in which certain options are incompatible with each other. For this reason, users outside the development group can be unaware of how the program runs in detail. The opportunity can be lost to make the software readily available outside of the laboratory of origin. This is a long-standing problem in scientific programming.

Rappture, Rapid Application Infrastructure [1], is a new GUI development kit that enables a developer to build an I/O interface for a specific application. This capability enables users to work only with the generated GUI and avoids the problem of the user needing to learn details of the code. Further, it reduces input errors by explicitly specifying the variables required.

Zori, a quantum Monte Carlo (QMC) program, developed by the Lester group at the University of California, Berkeley [2], is one of the few free tools available for this field. Like many scientific computer packages, Zori suffers from the problems described above. Potential users outside the research group have acquired it, but some have found the code difficult to use. Furthermore, new members of the Lester group usually have to take considerable time learning *all* the options the code has to offer before they can use it successfully. In this paper we describe the use of the Rappture toolkit to generate a GUI, labeled Zopi (Zori Processing Interface), for the Zori computer code.

Zori Basics

Zori is a computer code that implements the QMC method for the electronic structure of atoms and molecules. A detailed description of the structure and capabilities of Zori, as well as the fundamentals of QMC can be found elsewhere [3-8]. We present a general description of the steps a typical QMC calculation below for understanding the complexity of the method and how the GUI addresses this process.

A QMC electronic structure calculation is facilitated by an initial wave function Ψ typically generated by an external quantum chemistry package. For this purpose, the geometry of the molecule to be studied is specified followed by determination of Ψ_T, where the subscript labels the particular wave function for the specified geometry. Examples of programs frequently used for this purpose are GAMESS [9] and ADF [10]. Output files generated by these programs must then be transformed into Zori input format. A QMC calculation stochastically probes configuration space with entities called walkers guided by Ψ_T. Each walker represents the positions of all the particles of the system. The initial set of walkers is generated at random. Next the algorithm can proceed along two paths: Variational Monte Carlo (VMC) or Diffusion Monte Carlo (DMC). Each of these is a Monte Carlo method. Since the latter gives more accurate results, but it requires greater computer time than the former, QMC researchers tend to make first a VMC run to relax the initial configurations and then to optimize the parameters of Ψ_T followed by a series of DMC calculations to obtain statistically useful results. In ZORI the optimization of Ψ is performed by a module called ZOPT, which links to the optimization code Opt++. [11]

The way Zori works is straightforward, but the input files can be confusing. Xml formatted files are used to input information. These files contain information on execution time, algorithm type, number of walkers per processor to be used, etc. However, different options at execution useful for code development require parameters to be specified that are not pertinent for applications. Moreover, users are typically not fully aware of all options or the variables needed to be determined. In addition, some input files contain variables that are not usually modified other than for method development purposes, that adds an extra level of complexity and possibly confusion for the user.

Zori outputs two files. Periodically, i.e., for some fixed number of MC iterations, a new walker file is written to disk with the most recent positions of the walkers. With either DMC or VMC runs, Zori continuously updates a binary file called zori-energy.out, which contains the energy list of all the positions probed by the walkers. Because MC calculations normally run for long periods, it is customary to probe periodically how the energy fluctuates during a run. A tool included with Zori called Zavg takes the list of energies from the file, calculates the mean, variance and statistical error and saves these quantities in a text file, which can be easily visualized with programs such as GNU-Plot [12].

Note that Zori documentation can be found at *www.zori-code.com* . This web site also includes a Wiki page for submitting questions and problems related to Zori.

A look into Rappture: Wrapping applications

Documentation of Rappture is readily available on the NanoHUB website, www.nanohub.org. Also accessible at this address is a complete audiovisual tutorial for the toolkit. Furthermore, Rappture is also supported by a wiki page from which the program can be downloaded.

Rappture is a powerful tool that can be used at different levels of expertise. It can be utilized as a wrapping tool, if the target program has already been created, or it can be incorporated directly into the application.

Rappture has the special flexibility of being able to link to programs written in multiple computer

languages. The interface between Rappture and the application program can also be written in various languages including C, Python, Tcl and Mathlab.

Rappture comes with two libraries: one with unit conversions and the Rappture Library, which contains the objects necessary to develop the GUI for the program. In practice, in order to run Rappture, two simple files must be constructed.

The first one is an xml file, invariably called tool.xml, which contains all the imagery necessary for developing the GUI. There are several XML elements that can be used in the construction of the GUI including strings, meshes, clouds, images, and options; among others. Use of this XML format is extremely simple, and again, the Rappture Wiki page offers extensive aid on how to do it.

The XML file also contains a link to the second file, called the driver, which is the file that interprets the XML elements and turns them into the GUI. It also takes all the information given by the user in the GUI and sends it as input to the application.

A general example of a tool.xml file follows.

```
<?xml version="1.0"?>
<run>
 <tool>
  <about> Welcome to ZoPI</about>
  <command>python @tool/./zopi.py @driver</command>
 </tool>
 <input>
  <boolean id = "rn_psi">
   <about><label>Yes or No?</label>
   <description>Please choose</description>
   </about>
   <default>no</default>
  </boolean>
 </input>
 <output>
  ...(other Rappture XML elements)
 </output>
</run>
```

This example demonstrates generally how the tool.xml file should be structured for a given application. It starts by opening the main element - the run instruction - followed by opening the tool command. The location of the driver file is then specified in the command line. In this case, our driver file, zopi.py, is located in the same directory as the tool.xml file, which is the recommended option. Then, all the elements that need to be displayed by the GUI as input are now specified. We show the declaration of a Boolean as an example. Finally, the output elements that will appear in the output are described. These can include graphs, clouds and meshes among others.

The second file, called the driver, can be either a script that works as an interface between Rappture and the application “wrapping” it or it can be immersed in the application itself. This file can be coded with a number of programming languages, which gives Rappture increased flexibility for scientific

programmers who are accustomed to a particular language. In the next section a brief description of the file generated for the development of ZoPI, the Zori Processing Interface, is presented.

Development of ZoPI, the Zori Processing Interface

Learning how to use Rappture was not particularly difficult, which is one of the advantages of the toolkit. However, we needed to adapt Rappture's capabilities to Zori's needs. For instance, Fig. 1 shows a common set of steps that a user would follow using Zori.

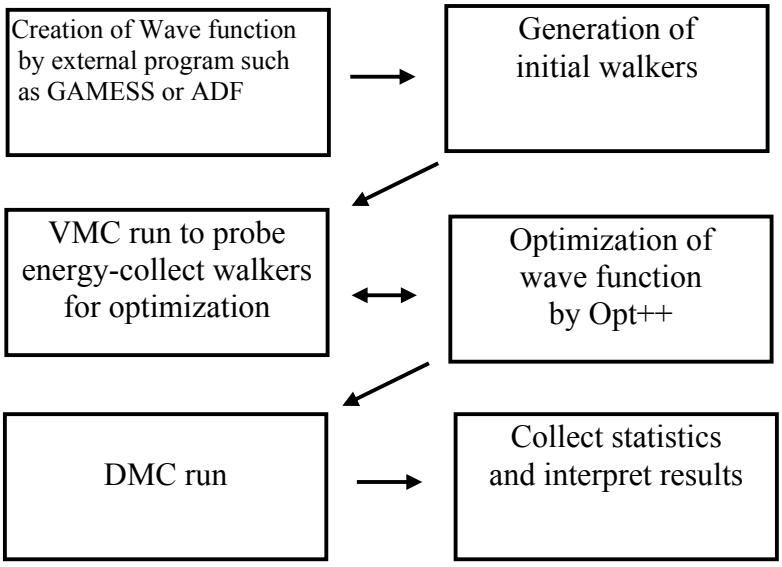

Figure 1. Flow diagram of a Zori run.

It is necessary that ZoPI include a linear structure composed of separate sections that follow this algorithm. The particular order of the steps should be evident as a way to guide the user easily into the correct usage of the package. Furthermore, QMC is well-known to have the benefit of being embarrassingly parallel, so it is common for users to run applications on more than one processor, on either local clusters, or on supercomputers, which means that Zopi needs to have an additional section to control the necessary options for generating the proper scripts for submitting the job to the queue system of the given computer. The submission of a job to a single processor is clearly also desired, so Zopi must also contain this capability.

Rappture offers a system that includes a tab interface in which each tab controls a step of the procedure. Note that the last tab addresses the submission section. Furthermore, the possibility of running all of the steps, i.e., generation of walkers, the VMC and the DMC runs, sequentially or one at a time is also included. Figure 2 shows what Zopi sees at the beginning of a run.

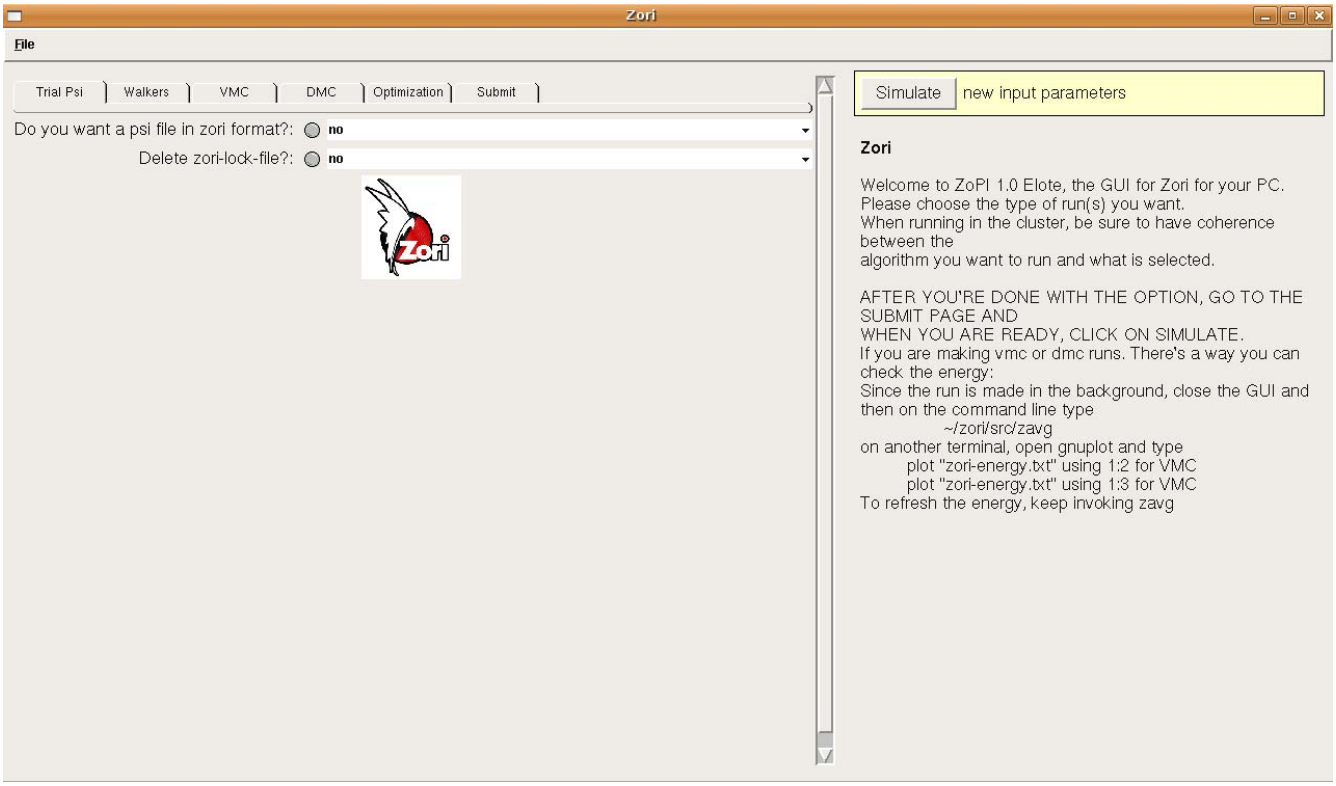

Figure 2. Initial Layout for Zopi. Run in Ubuntu 7.04.

In this figure, it is presented how the left side is in charge of the input. One can notice the five tabs:

- *Trial Psi*, for writing the information of the wave function to be used into Zori format.
- *Walkers*, for creating an initial set of walkers.
- *VMC*, for driving a VMC run.
- *DMC*, for driving a DMC run.
- *Optimization*, for calling OPT++ to optimize the correlated wave function.
- *Submit*, for taking care of submitting the job to a cluster or running interactively on a single processor.

The development of tabs through Rappture is specified in the tool.xml file. We present an example of the corresponding instructions.

```
<group id = "WT">
  <group id = "PSI">
    <about><label>Psi</label>
     <description>Creation of the wave function and orbitals .xml files</description>
    </about>
    ...(Rappture XML elements)
  </group>
 <group id = “CW”>
   ...
 </group>
 ...
</group>
```

The first group id is to encapsulate all the tabs. The second group id is for the first one, Trial Psi, and in it, one prompts the rest of the input this section requires. The list can continue, and the length of the tabs is automatically changed.

Although similar in structure, it is important to look at other tabs to gain insight on one of the advantages of Rappture, namely, compartmentalization. Figure 3 shows a section of the DMC tab. There are some other options in the Random Walk section that are not shown in this display.

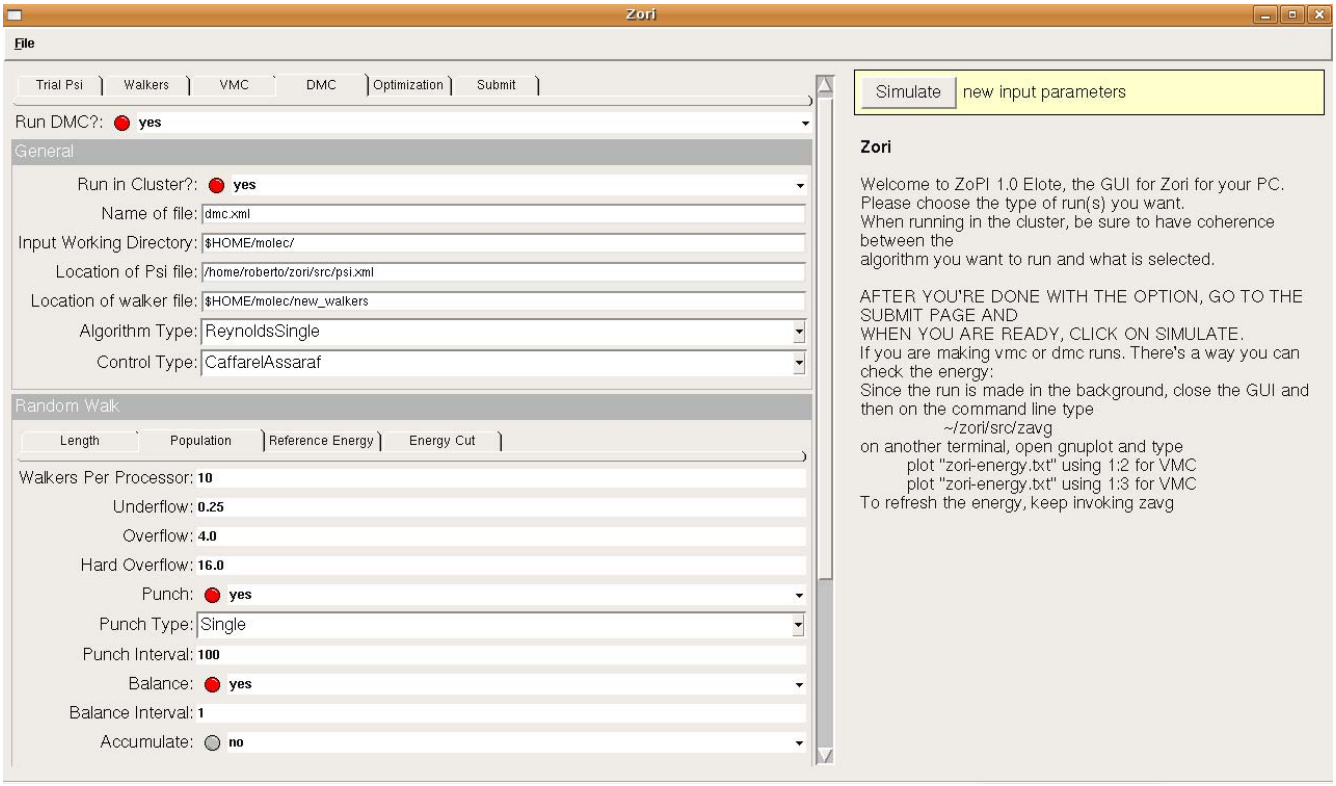

Figure 3. Layout of the DMC tab.

In this display one notices three sections: the run instruction which activates this section, the General sub-section which specifies the path for the input files needed and the type of DMC algorithm to be used and finally there is a Random-Walk section. In the latter, there appears to be more tabs, all of which must be filled out completely with the required information. Each algorithm type will prompt only the input it requires, making it easy for the user to know exactly which parameters he/she needs to specify. This has the additional advantage that users have a good idea of what they are filling out, and therefore can acquire further insight on the usage of Zori and, in general, of QMC.

The OPT++ tab enables one to run an optimization algorithm for the parameters of the wave function. OPT++ is added to Zori to improve its performance. This tab functions similarly to the others. The only difference is that it will call the module Zopt as opposed to the main Zori program like the rest of the tabs. This will be evident in the command line itself, which we describe in the next section.

As mentioned above, Zopi has an extra tab providing the option to run QMC calculations on a computer cluster. The design of this tab favors users at UC-Berkeley and the Lawrence Berkeley National Laboratory (LBNL), since it explicitly includes the typical specifications for clusters present

at these institutions. However, this program also provides prompts for a custom cluster script. Furthermore, Rappture's flexibility makes possible a rapid adaptation of Zopi, that permits the creation of a customized template for the cluster of choice. As stated earlier, one needs only to edit the xml file, and afterwards edit the zopi.py where the actual command stands. For instance, for the cluster of the Lester group, we created the following function that asks the user for the needed information, such as the number of processors to be used, the location of the executable of Zori, etc.

```
   def rn_gn(no, run, cmd, ld_lib_p,):
      cl = open('ganita.scr','w')
      cl.write('#!/bin/bash\n')
      cl.write('cd ' + os.getcwd() +'\n')
      if no == 1:
         cl.write('lamboot\nsetenv LD_LIBRARY_PATH ' + ld_lib_p  + '\n' + run +' '+ cmd
+'\nlamhalt\n')
         cl.close()
         exec_str_clus = 'qsub ganita.scr'
         os.system(exec_str_clus)
       elif no > 1:
         nu = str(no)
         cl.write('mpirun -x ' + ld_lib_p + ' -np ' + nu + ' ' + run + cmd)
         cl.write('\n')
         cl.close()
         str_clus = 'qsub -V -pe lam ' + nu + ' ganita.scr'
         os.system(str_clus)
      else:
         cl.write('Error in number of processors... Terminating...\n')
         cl.close()
      os.remove('ganita.scr')
```

The function writes a script called ganita.scr with this information, which is then sent to the Ganita queue system by the GUI. [Ganita is the name of the Lester-group cluster.] Zopi prompts for all the variables the function asks for, so there are no worries that some information is missing. A simple function can be written by the users to create a template for their cluster of choice.

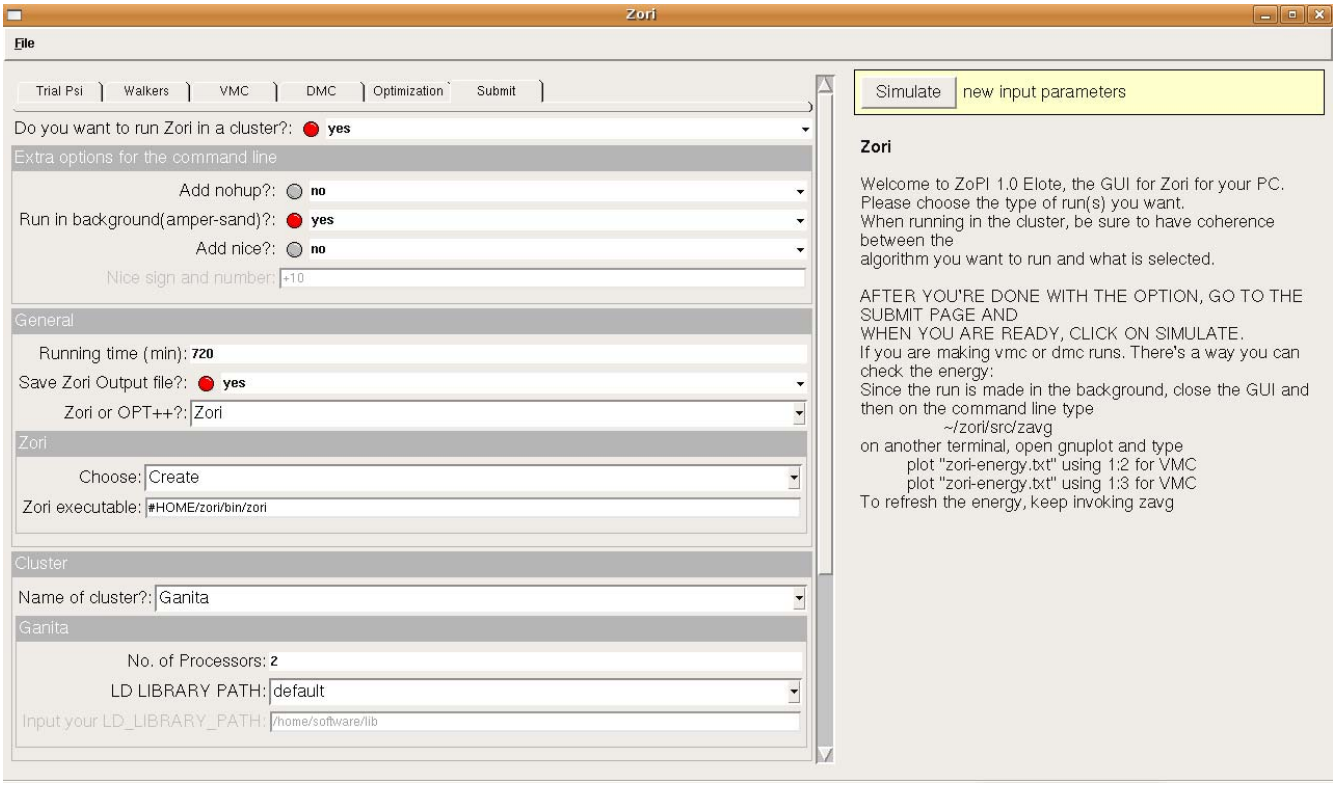

Figure 4. Cluster input tab.

Running Zopi

With the aid of the GUI, running Zori becomes quite easy to execute. The only thing the user needs to do is to click on the simulate button on the right side of the Interface after completing the requested information on the left side.

Internally the GUI saves the input values and is interpreted by Rappture based on the driver file information zopi.py. The Rappture Interface then builds the xml file necessary for running Zori. The interface then builds the appropriate command line to run Zori with all the specifications of the user. For instance, a typical Zori command line for a DMC calculation would look as follows:

./zori –i dmc.xml –p psi.xml –r new_walkers –t 720

Here we are assuming the executable for Zori, as well as the needed xml input files, are located in the directory from which we are sending the job. This command line asks Zori to run using the file dmc.xml as the main input file, and the psi.xml file for the wave function, the walkers file(s) comes from the new_walkers prefix and finally, Zori will run for a maximum of 720 minutes.

The optimization program, OPT++ has a slightly different command line. The main difference is that it invokes the module zopt instead of zori.

./zopt –i walkers.xml –p psi.xml –r new_walkers

All previous steps are now hidden from the user and performed by the GUI. As an example of how the

GUI would be used, let's say that a user has a valid wave function file that Zori can understand, usually called psi.xml and that the user wants to run a VMC calculation; he or she would only have to go to the VMC section of the GUI and activate this calculation by selecting yes in response to the Run VMC question. This will change the look of the GUI, displaying the fields for the information one needs to specify for a VMC calculation. After filling out this information, the user need only go to the Submit section of the GUI to specify whether the calculation is to be run interactively on one processor or to the queue system of the cluster where the GUI is running. After providing the information the user would click on the simulate button on the right to perform the calculation. If the run is chosen to be performed interactively the right panel would then show the output of Zori. Figure 5 shows a screen shot of the GUI during a VMC calculation. At this point the user can choose to close the GUI and let the calculation continue running in the background or continue to monitor the progress of the run by calling Zavg.

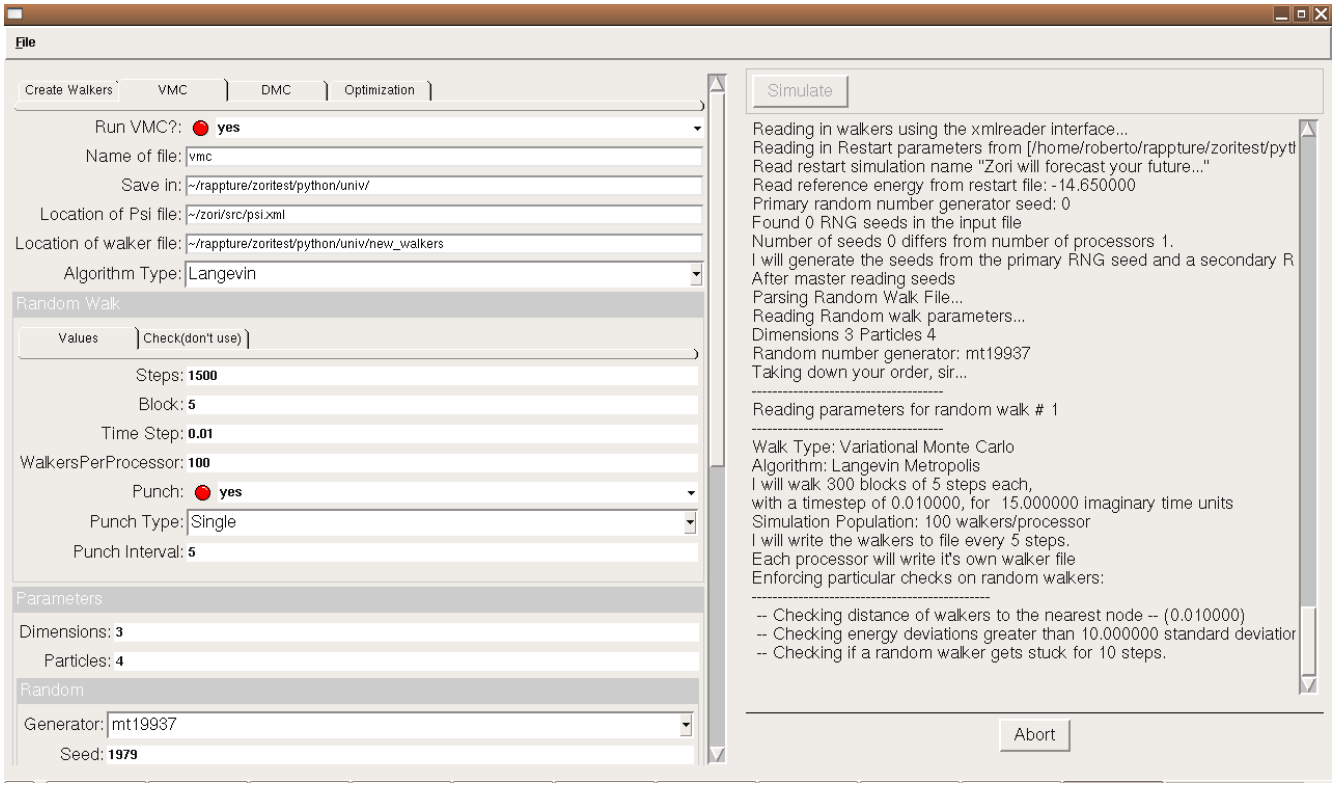

Figure 5. VMC run. Snapshot of a run taking place and has not yet ended.

When the calculation is finished, Zopi outputs a simple indication of which runs have taken place. As stated before, the user can choose to close the GUI after submitting the job. This message will only appear if the GUI is still active when the run ends.

Conclusions

As has been shown in many areas, the existence of a simple and understandable way of running a given computer package is essential to making the software accessible to the general user. In scientific programming this capability is often not given high priority. Increasingly production-package developers are turning to the development of a GUI to make their application codes more user-friendly.

In this paper we have presented the use of Rappture as a simple and reasonable way of generating a good, clean and organized GUI for a quantum Monte Carlo (QMC) application code called Zori. The resulting GUI successfully absorbs all the complexity of the input necessary to run the application in an easy to understand and organized way. It makes it clear to the user not only the usual set (?) and order of steps to be followed, but also the various alternatives available to perform each of them. For each section, it states specifically the variables to be specified, giving a simple explanation of them and, in some cases even suggested values. The resulting GUI also retains flexibility with the option of separately submitting each step of the calculation. In some calculations, several VMC relaxation-optimization cycles are necessary to get a good wave function for a DMC run.

With the development of a simple GUI for Zori called Zopi, the authors hope to broaden the use of the very powerful quantum chemistry method, QMC, and to show to the scientific community the ease and benefits of generating such tools. The latter include instructional advantages for students in institutions of higher education.

Acknowledgments
R. O.-A. and C. A.-B. thank DGAPA-UNAM for financial support during the semester at Berkeley. R. S.-F. and W. A. L. were supported by the Director, Office of Science, Office of Basic Energy Sciences, Chemical Sciences Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. The authors give special thanks to the rappture mailing list members, Michael McLennan and Alan Aspuru-Guzik for invaluable help during the development of this project.

References
[1] Lundstrom, M., Klimeck, G., "The NCN: Science, Simulation, and Cyber Services." 2006 IEEE Conference on Emerging Technologies - Nanoelectronics. pp. 496-500
[2] A. Aspuru-Guzik, R. Salomón-Ferrer, B. Austin, R. Perusquía-Flores, M. A. Grifin, R. A. Oliva, D. Skinner, D. Domin, and W. A. Lester Jr., J. Comp. Chem. 26, 856 (2005).
[3] B. L. Hammond, W. A. Lester Jr., and P. J. Reynolds, Monte Carlo methods in ab initio quantum chemistry. (World Scientific, Singapore, 1994).
[4] J. B. Anderson, Quantum Monte Carlo: Atoms, Molecules, Clusters, Liquids and Solids. In Reviews in Computational Chemistry, Lipkowitz, K. B., Boyd, D. B., Eds.; Wiley-VCH: New York, 1999; 13, 132-182.
[5] M. Foulkes, L. Mitas, R. Needs, and G. Rajagopal, Rev. Mod. Phys. 73, 33 (2001).
[6] A. Aspuru-Guzik and W. A. Lester Jr., Quantum Monte Carlo methods for the Solution of the Schrodinger equation for molecular systems in Computational Chemistry. In Computational Chemistry, Le Bris, C. Ed., Handbook of Numerical Analysis. Elsevier: Amsterdam, 2003; X, 485.
[7] A. Aspuru-Guzik and W. A. Lester, Jr., "Quantum Monte Carlo: Theory and Application to Molecular Systems," Adv. Quant. Chem. 49, 209 (2005).
[8] P. J. Reynolds, D. M. Ceperley, B. Alder, and W. A. Lester Jr., J. Chem. Phys. 77, 5593 (1982).
[9] M. Schmidt, K. Baldrige, J. Boatz, S. Elbert, M. Gordon, J. Jensen, S. Koseki, N. Matsunga, K. Nguyen, S. Su, T. Windus, M. Dupuis, and J. Montgomery, J. Comp. Chem. 14, 1347 (1993).
[10] G. Velde, F. Bickelhaupt, E. Baerends, C. Guerra, S. V. Gisbergen, J. Snijders, and T. Ziegler, J. Comp. Chem. 22, 931 (2001).
[11] J. C. Meza, P. D. Hough, And P. J. Williams, OPT++:An Object-Oriented Nonlinear Optimization Library, 2004, Internet site: (http://csmr.ca.sandia.gov/opt++/).
[12] Williams T., Kelley C., http://www.gnuplot.info/.